Vol 3 Issue 9 Oct 2013

ISSN No : 2230-7850

Monthly Multidisciplinary Research Journal

Indian Streams Research Journal

Welcome to ISRJ

RNI MAHMUL/2011/38595 **ISSN No.2230-7850**

Indian Streams Research Journal is a multidisciplinary research journal, published monthly in English, Hindi & Marathi Language. All research papers submitted to the journal will be double - blind peer reviewed referred by members of the editorial Board readers will include investigator in universities, research institutes government and industry with research interest in the general subjects.

Address:-Ashok Yakkaldevi 258/34, Raviwar Peth, Solapur - 413 005 Maharashtra, India
Cell : 9595 359 435, Ph No: 02172372010 Email: ayisrj@yahoo.in Website: www.isrj.net

Indian Streams Research Journal
Volume-3, Issue-9, Oct-2013
ISSN 2230-7850

Available online at www.isrj.net

Assessing The Software Architecture Toward Evolution

R. Aroul Canessane And S. Srinivasan
Research Scholar, Sathyabama University, Chennai
Professor & Head, Dept. Of Computer Science and Engineering, Anna University , Madurai

Abstract:The architectures of software system confines non functional requirements (NFR), hence the decisions that are taken at the time of creating architectural design have a major impact on resulting system. We have proposed a design methodology for the architecture, which uses an iterative method for evaluating and transforming the software architecture until the NFRs are satisfied. The evaluation is carried out by means of scenarios, reasoning, mathematical modelling and simulation. The transformations are carried out by imposing certain architectural style, design patterns, conversion of NFR to appropriate functionality and distributing the NFR's.

Keywords: Architectural transformation, fault tolerant, performance, interfaces.

I.INTRODUCTION

One of the complex activity is the conversion of the requirements specification into corresponding software architecture for the application. Though the other phases of activities are also challenging, they overcome them with the methodological approach, procedural and technological support to the software engineers. The design phase lack in craftsmanship and formalized procedure like other phases. The software architecture domain has been creating an attention by the researchers in the recent years, because of the NFRs which are influenced by software architectures of the system. The design that we have may create bottleneck for the performance and the reliability for the system. The job of the software engineer is to balance all the requirements while he makes a design.

In this paper we have presented a design methodology for providing a support for the design process for making balanced and optimized NFRs. We have defined an iterative method to access the degree of architecture, which supports all the NFRs and refines the architecture until the NFRs are complimented. The method that we have proposed complements the traditional methods of designing that focuses on NFRs rather than the functionalities.

The paper is organised as section 2. Requirements Engineering, section 3.Method overview, section 4.Architecutural design functionalities, 5.Non functional requirements, section 6.Architectural transformation, section 7.Result and Discussions and section 8.conclusion.

REQUIREMENT ENGINEERING

The purpose of requirement engineering is to identify and specify the requirement, it not the main aim of us, instead we have used the requirement specification as an input to the architectural design and we have established different terminology for different requirement concepts. System requirements lead the top level of the requirements which is a combination of software, mechanical requirements and the hardware. In this paper we concentrated on only the software requirements and we have ignored other categories of requirements. The software requirements are classified as functional and non functional. The different NFRs are stated as performance, interface, operational, resource, verification, acceptance, documenta tion, security, portability, quality, reliability, maintability and safety requirements (IEEE-std 830). These NFRs are also called as attributes of system properties. Functional requirements are highly related to the domain functionalities of the application. The functionality requirements are deployed suing the subsystem or components of the software, whereas the NFRs can be grouped into developmental NFRs which determine the quality of a system with respect to the software engineer perception and operational, e.g. maintainability, flexibility, demonstrability and reusability. The operational NFRs are defined by the system operation, e.g. performance, fault tolerance, robustness and reliability. NFRs cannot be easily pinpointed as functional requirements are done.

The example that we have taken in this paper is based on the experience of the SmartHome. Though a SmartHome (home automation) system consists of different types of sensors, presentation devices, alarm bells, communication devices, user interfaces, we move with simplest functionality behaviour of abstracted system.

Figure1. Input / output view of SmartHome system

The abstraction of the SmartHome figure 1.consists of an abstracted set of inputs and a set of outputs which represents numerous numbers of sensors and its corresponding indicators. Based on the outputs they behaviour are implemented using the process for the inputs. NFR of the safety home system are the worst case in response time for making an alarm, reliability, availability and efficiency.

METHOD OVERVIEW

Usually NFRs are dealt with a very informal and non-methodical approach in the software architectural design. Tests are used to determine to check whether NFRs are fulfilled after the implementation. If it is not fulfilled the part of system is considered for redesign. The system architects are often building systems in the domain, the experience of it helps them to minimise the system redesign. The research on software engineering has spent some effort in several NFRs, e.g. object oriented system research has improved in reusability and some of the real-time system study has helped in developing NFRs. They concentrated on single NFR and they lack in addressing the combination of different NFRs. Only a single NFR is not present for a real time system, it has to achieve multiple NFRs. For example most of the real time system must be reusable and it must be maintainable to achieve the cost effectiveness, whereas the fault tolerant systems must also fulfil the requirements such as maintainability and timeliness. No such pure fault tolerant, reusable, real time and high performance system exist, though the researcher project many artefacts of pure real-time systems. All the realistic and practical computing system must satisfy all the NFRs, however constructing such a system is hard, because of the NFRs conflictions. Such as reusability and performance, fault tolerant and real-time computing is contradicting.

Conventional design focuses on the system functionality and they do not concentrate on non functional requirements. However so many systems that they have developed concentrate on single NFR and they lack in treating other NFRs. They have provided a secondary importance for the other NFRs. We consider those approaches as unsatisfactory, because software engineers have to balance the NFRs for the realistic system.

Method

The method begins by taking the requirements specification as an input to the method and producing an architectural design as an output. This is the first version of the design which is reduced in subsequent phases. The steps of the methods are represented diagrammatically in the figure 2. The process is initiated with the architectural design with the functions that are specified by the requirement specification, the NFR are addressed explicitly at this stage though the software engineering will not design the system which lack in reusability and reliability. The output of this is a first version of the application architecture. The architecture is first evaluated by considering the NFR. The NFRs are given estimated values and they are compared to the actual values of the requirements. If the estimations are good and are up to the mark the design process is stopped. Else it enters into the second stage for architecture transformation. At this stage the design is improved with the help of appropriate NFR optimizing transformations. Each transformation produces a new version of design which is given as a feed back to earlier stage. The new version is again evaluated, the process is iteratively repeated until the NFR values are fulfilled, until the software engineer decides there is no other feasible solution persists. One NFR based system follows iteration but only Smith considered performance.

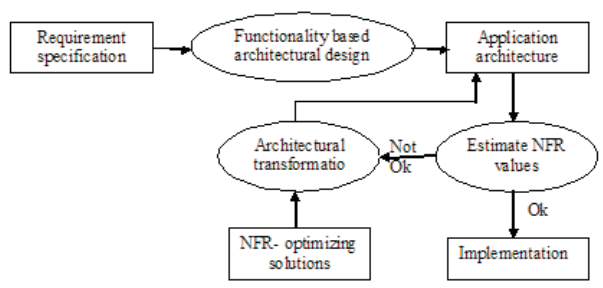

Figure 2. Architecture design method outline

The method presented in this paper was applied to SmartHome system. Our experiences with this project have shown us that the architectural design method does not restrict the creativity of the engineer but instead supports it throughout.

ARCHITECTURE DESIGN – FUNCTIONALITY BASED

The requirement specification is used to build the top level architectural design of the system. The basic task of this phase is to identify the key components or the core abstractions based on which the system will be structured. Though the abstractions are designed as objects, these modelled objects cannot be found in the application domain immediately, they are considered as the results of a design process which analysis's the domain entities, abstractions and models the architecture entities. As soon as the abstractions are recognized then the interactions between those abstractions are elaborated. Identifying all the entities and defining the architecture in object oriented design and using those entities they define the inheritance. In our experience it is not an acceptable thing for going to bottom up approach rather than going for a top down approach which deals with the detail information of the system. The architecture of the SmartHome system consists of basic entities that are devices and the controllers. But a SmartHome system consists of sensors different actuators such as alarm, sleep mode, messaging etc. The identified entities of the system are not straightforward and concrete entities of the architecture design covers the multiple domains.

NON FUNCTIONAL REQUIREMENTS

Identifying and evaluating the non functional property explicitly is the major characteristic of architectural design method, without having a complete system available. The traditional method in a software industry is to evaluate

the system after implementation and measure the values for non functional properties of the system. The disadvantage is the amount of effort that has been put on developing the system does not guarantee in fulfilling the non functional requirements. Several systems have been already developed for estimating non functional attributes during the development stage which leads to mishaps. It is not an easy to measure system properties based on the abstract specification of an architectural design. On cannot give a complete measure of NFRs for a system based on architectural design, rather the aim is to evaluate potentiality of the architecture that is designed and we can try to reach the required level of NFRs.

The architectural style that is chooses for a design cannot provide highly equipped system, each and every style has its own advantage and disadvantage. Four approaches are defined in this paper which identifies the non functional requirements they are scenarios based, simulation based, mathematical modelling and the objective reasoning.

Evaluation –Scenario based

Scenarios are created for accessing the NRF, which defines actual meaning of the NFR. Example, maintainability requirement can be defined by the scenarios which captures the typical modification in requirements, with respect to hardware. The scenarios can then be evaluated with the changes that are required to adapt the architecture for the appropriate situation. Robustness of the architecture can also be evaluated with respect to the invalid outputs.

Scenario representativeness deals with the effectiveness of the approaches done by scenario based. Accurate results can be got for the actual specified scenario. Object oriented method uses the use case scenario which specifies the system behaviour. Scenarios must be developed in two sets, one is for the design and another one is for evaluation. Once the architecture design version is ready, the software engineer has to run the scenario and results must be evaluated. At the most the change in scenario, is nothing but the reorganizing the architecture, which can conclude a low maintenance in architecture.

The statistical evaluation can be made on the scenarios by using the testing code of statistics, which define ratio between the successful scenarios and failed scenarios. The quality of the scenario is defined by the software engineer who is running the scenario. In our experience the scenario based evaluation is useful for the development of NFRs such as maintainability, which can be easily identified by changing the scenario. The SmartHome application many maintenance scenarios can be evaluated as shown in the table 1.

Scenario	Description	Direct/Indirect	changes
1	Detector	Indirect	Will require modification based on the usage.
2	Fire extinguisher	Indirect	The components must be modified with respect to the place we implement.
3	Hardware platform	Direct	To fulfill market expectations and the customer need the complexity must be defined.

Table 1. Scenario evaluation

The evaluation that is made based on the estimation of effort required for the new environment adaptation.

Simulation based

Simulation, not only used for evaluating the NFRs, it also help in evaluating the Functional requirement of a design. A simulation helps to define the interaction, behaviours and the functionalities which uncovers inconsistency in design and details the entities of the architecture.

Simulating the architecture by implementation of application architecture is our second approach to estimate the NFRs. Some components of the architecture are implemented and some of them are simulated which leads to a implemented system. The components which have been implemented can also be simulated with an appropriate abstraction. Hence the implementation can also be used for simulation with the application behaviour. After the simulation of the application has been completed the NFR – robustness can be easily evaluated by giving faulty input to the simulated application and find the tolerance among the architecture entities.

Simulation helps in evaluating the operational NFRs i.e., fault tolerance, whereas the simulation by changing the scenario, the maintenance can be defined by measure the effort. The simulation helps in our SmartHome example by defining the interface to the physical sensors and the indicators by using a layer of the software which is used in communication as show in the figure 3.

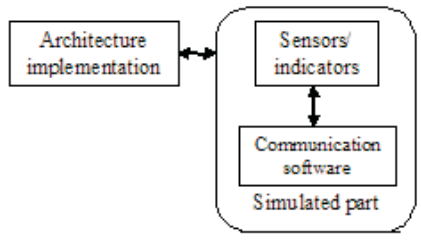

Figure 3.Simulation system

The communication and the sensor behaviour can be simulated, with the help of the interface along the top most level architecture. A Markov model can be used for evaluating the scenarios is show in figure 4. Using this markov model the robustness can be evaluated before the communication software is designed. The accuracy hence forth can be defined with respect to the real world implications. The parameter that has been used in the SmartHome system has been show in the Table 2.

Parameter	Value
(Un)Marshalling Requests	Constant 0.1*s*
(Un)Marshalling Responses	Exponential, mean=5.0*s*
Request Transmission Times	Exponential, mean=1.0*s*
Response Transmission Times	Exponential, mean=10.0*s*
Request Computation	Exponential, mean=0.1*s*
Frame Encoding Time	Exponential, mean=20.0*s*
Display time on the PDA	Exponential, mean=20.0*s*
PDA Speedup factor	0.2
PC Speedup factor	10.0
Processors Sched. Policies	FIFO

Table 2. Parameters that has been used in the SmartHome

system example.

Mathematical modelling

Research communities actively participating in various fields, like high performance computing, real-time systems etc., have developed plenty of mathematical models that may be used to evaluate and assess the operational NFRs,

The mathematical models enable us to perform static evaluation and this is where it is different from the other approaches. For example, when we engineer high performance computing systems, mathematical modelling can be used to evaluate the different application structures and decide on the one which provides maximum performance.

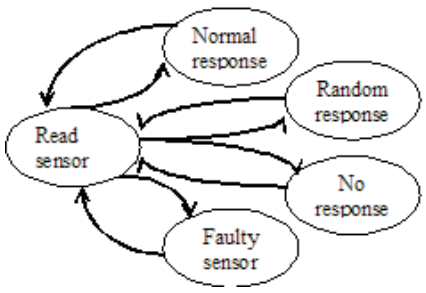

Figure 4. Markov model for communication and sensors

Mathematical modelling and simulation are alternatives to each other as both methods of evaluation have the basic idea to assess operational NFRs. But, in some instances, these two approaches may be combined. For example, one may use mathematical modelling to estimate the computational requirements of individual entities in the design. The simulation then uses these results to estimate computational requirements of the different sample execution sequences of the architecture.

In the SmartHome example, we have an alarm system, using the mathematical modelling we can relate to one of the NFR, as the worst care response time for the alarm is 3s. Assuming the model that is shown in figure 5 which has a poling system were the inputs are evaluated periodically. One sensors evaluation is done by sending the message to physical sensor since they have a communication software, waiting time, receiving time of response and evaluate the alarm conditions. It this performance model example, we assumed

Send request needs - 2ms
Receive answer needs - 2ms
Evaluation needs – 5ms
Communication needs – 8ms

If the system needs around 150 sensors, we used a round robin method for polling. the worst case of response time can be roughly estimated as

150 * (2+2+5+8) = 3.3s.

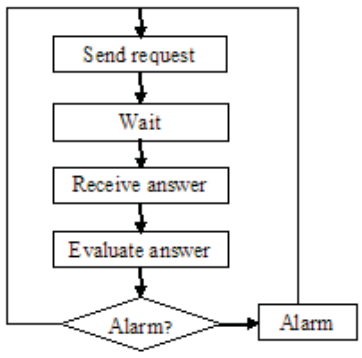

Figure 5 Model for sensor value reading

Objective Reasoning

The fourth approach to access the NFRs is via reasoning using the logical arguments as a base. Software engineers frequently have insights that may prove extremely valuable and therefore helpful in avoiding the bad decisions on design, which comes out of their experience. Most of these can often be explained by reasoning which is logical in nature, even though some portions of these are based on the previous evidence.

The difference from the other approaches in this is that the assessment process is more implicit and is based on less objective factors such as experience and intuitive understanding. This does not mean that this approach is not as usable. Software architects we interacted had, well-developed ideas about 'good/bad' designs.

These architects always started the problem with the intuition of there is something wrong. Based on that intuition they have made the approach of logical reasoning, for example an experienced engineer can easily identify the maintainability problem in architecture, by redesigning the scenarios he can rectify the major problem easily.

The fire alarm which is used in the SmartHome application is concurrently inheritable; hence we choose the concurrency model. When we tried to use pre-emptive scheduling we are not able to consider the race conditions. Hence we were implementing a round robin which makes a line of reasoning that to avoid the architectural transformation. The round robin transition is shown in the figure 6.

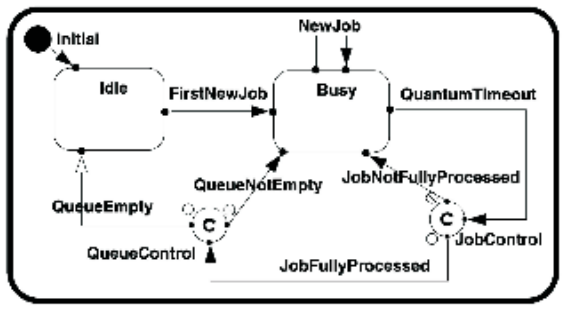

Figure 6. Round Robin State chart

ARCHITECTURAL TRANSFORMATION

After the assessment of the architectural design properties are completed, the estimated values are then compared to requirements specification. In case some non-functional requirements are not satisfied, changes must be made to the design to incorporate those requirements. The software engineers must check the results of the various evaluations and identify what the flaws that exists in the design. Usually the evaluations offer enough hints about where the rectification is required in design by giving low scores while making the evaluation.

The evaluation of the NFRs is performed with respect to a certain context usually, for example, a GUI system or database or hardware. If the NFR is not satisfied, we can change the context of the design or sometimes it may lead to a major change in complete architecture. In this paper we have discussed about the architectural transformation. We tried to create a new version each time which fulfils the functionality, but the change in values for the properties.

The consequence that is faced by the architecture transformations is, most of the time the transformations affect properties of the architecture, some may be positive and some may be negative for the properties. For example some of the strategy pattern which is used for design increases the flexibility in class with respect to the behaviour. The performance may be reduced since one class object invokes other objects of strategy instance which defines the behaviour. Thou in this case a positive effect have been increased by a minor impact on the performance.

In this paper we have discussed about the five categories of architectural transformation, which has been organized in the decreasing impact in software architecture. The steps that are carried out are:

1.Impose the architectural style.
2.Impose the architectural pattern.
3.Impose a design pattern.
4.NFRs to the functionality conversion.
5.Requirements distribution.

A single transformation is not possible to solve all the NFR, at least two or more transformation is required. Each category has been discussed in below in more detail.

Imposing architectural style

Shaw and Garlan have presented many architectural styles that enhance the system for certain NFRs but are less supportive for the other NFRs of styles like the layered style, generate better flexibility by adding several layers of abstraction, but usually end up by decreasing the performance of in resulting system. Every architectural style is best suited to a particular set of non-functional requirements as a system property. Since such a transformation completely changes the structure of a system, the style must be chosen carefully.

It is usually not possible to merge multiple architecture styles together, but different styles can be applied at different levels of the system, for example, at the system level and at the sub system level. As long as the subsystem that has a style different from the system and functions correctly at its level, it is applicable to use another architectural style. In our approach, we have tried to differentiate explicitly the components that provide the functional requirements and the system structure which decides the NFRs.

In real world scenarios, such an explicit differentiation cannot be made since various parts of the former may influence several system NFR properties like robustness and reliability. An architecture style model of simple function is shown in figure 7.

An assessment of efficiency and the performance of the system will conclude that the design is not adequate since all the Outputs have to check the state of all the Inputs. Deviations may be added as a method of checking only those Inputs that have been changed to a value other than which is in the acceptable range. Every Input must create a Deviation and store them in a common place. Now Outputs must only check that particular place and the functions accordingly to investigate the behaviour. The solution is identified by using the Blackboard architectural style as shown in figure 7.

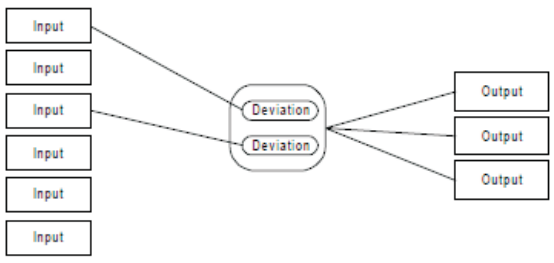

Figure 7. Black board style architecture for fire alarm system

Imposing architectural pattern

The next category in transformation is that of applying an architectural pattern. A pattern is different from a style; this pattern affects a larger part of the architecture. Certain rules are followed in the architectural pattern such that they specify how the system will deal with an aspect of the functionality such as persistence or concurrency.

The fire alarm system in the SmartHome application could serve as an example for the concurrency behaviour. The functional view of a system as shown in figure 1 assumes that the reading of the inputs and produces an output takes concurrently with the help of a thin pre-emptive thread and these solutions can be checked for the reliability and efficiency. These threads are error-prone, when accessing the shared data these pre emptive threads may cause the racing conditions. Therefore to solve this problem of reliability, we have used a pattern called as periodic object, it provides appropriate granule of concurrency. This periodic object can be defined as abstract object which is activated with the help of scheduler object tick method. The subclasses deploy the tick method of their own that defines the slicing and execution of active object periodically. The slicing of thickness defines the degree of granularity and concurrency. The result of all Tick methods must be returning a value within the predefined time. Design pattern and design rule that has been given in the example influences the entire architecture because of the

effectiveness of outputs are determined by the inputs. We have induced an algorithm in the figure 8 which handles the longest worst case of response times.

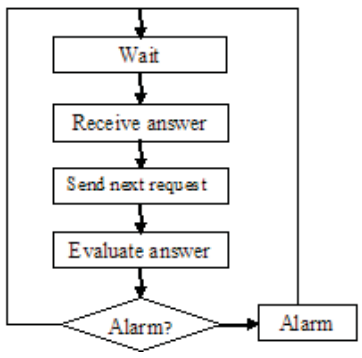

Figure 8. A modified model reading sensor model

Such a high-level transformation of the algorithm causes coupling among the Inputs, it needs simultaneous operation where the data is evaluated, the same time when another physical sensor is operating in the alarm system. Therefore, the entire architecture is affected. Applying the same mathematical models previously, however, the new worst case response time is.

150*(2+2+8) ms = 2.3s

Imposing Design pattern

This transformation is a far safer one, in the fact it is less dramatic. For example, an abstract factory pattern can be introduced to abstract the process of initialization of its clients. The abstract factory pattern increases the maintenance, flexibility and the extensibility of system, because it encapsulates the type of actual classes that are instantiated. But it will decrease the efficiency by creating the new instances because of the additional computations which reduces the performance and the prediction. Like imposing the architectural style which needs a complete reorganizing of the architecture is not needed in the imposing of design pattern because it deals on the basis of subsystem. Hence forth limited number classes will be alone affected. In the fire alarm of SmartHome, the evaluation of change in scenario results that the inputs that has to be given to the hardware, those hardware parts needs some changes for the NFR maintainability. The behaviour is defined by the standard of the actual sensor's used in the application, hence we have used a pattern called as point pattern here the input device is separated from the input point as show in figure 9.

NFRs to functionality conversion

NFRs can be converted to functional requirements and this is the next transformation that we have discussed. This extends the functionality of the architecture (not in the problem domain) but helps to identify a NFR. Exception handling is technique which helps to add functionalities to the components in order to increase its fault-tolerance.

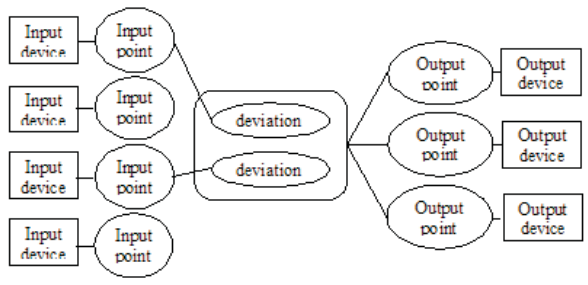

Figure 9. Improve the flexibility of system by point pattern

In our example of the SmartHome, alarm system for fire, self-monitoring and availability can be defined as NFRs.

In some cases faults has to be handled using the redundancy of the hardware, whereas other NFRs must be indicated to the system maintenance personality. Similar to alarm requirements, rest of the requirements can also be identified by transforming to the functional requirements. We have shown the architecture of the self monitoring in the figure 10.

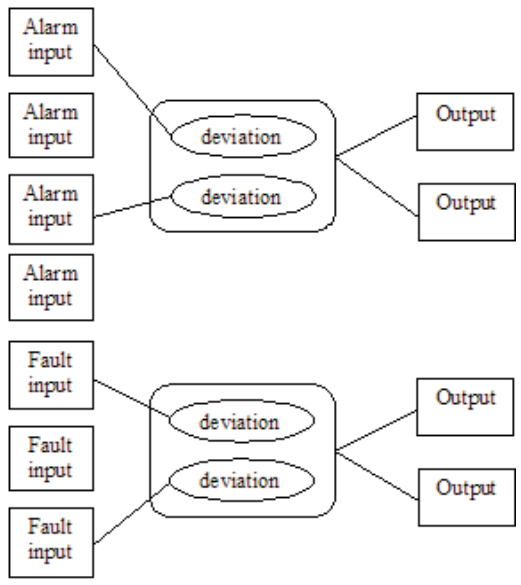

Figure 10. Self monitoring transformation to functional behaviour

Requirement distribution

This requirement distribution type of transformation takes care of fulfilling NFRs using the divide-and-conquer principle: an NFR which exists at system level is divided and assigned to subsystems or subcomponents that together form the system. Hence, an NFR X is divided into n different components that together make up the entire architecture, by assigning an NFR xi for each component ci hence $X=x_i + ... + x_n$.

Another approach is to distribute the requirements by dividing an NFR into many function related NFRs. For instance, in the distributed systems, the fault-tolerance can further be subdivided into fault-tolerant computation and

also fault-tolerant communication.

The alarm system for fire which is present in the SmartHome is implemented as distributed systems, where a CPU-based system controls the entire building. Systems of similar type communicate with each other and if an alarm is given by one system, all other systems indicate the same. Such an indication can be achieved by having a copy on the blackboard style available in all the subsystems. This way of distribution must be done using the softwares that are used for communication at the lower level layer of software operation, which assures the consistent copies of blackboard distributed in all the systems. The diagram of this architecture is shown in figure 11.

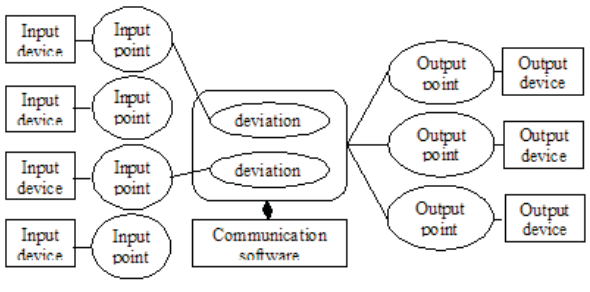

Figure 11. Distributed aspects in fire alarm system

Hence, the NFR which mentions how well the alarm system must deal problem related to communication is assigned to soft ware's that are used for communication, which helps in a distributing a system level requirement to a component.

RESULTS AND DISCUSSIONS

We have implemented the evaluation and transformation procedures using UML2.4.1, we found a drastic change in the identification of NFRs which have supported our research work. We have plotted the comparison results of the NFRs identification with an existing method in the figure 11. After the transformation of the architecture we have identified the betterment of NFRs. In the SmartHome application, which consists of many components, such as sensors, alarms etc. The NFRs have been identified with respect to those components. We have shown around seven NFRs in the figure 12.

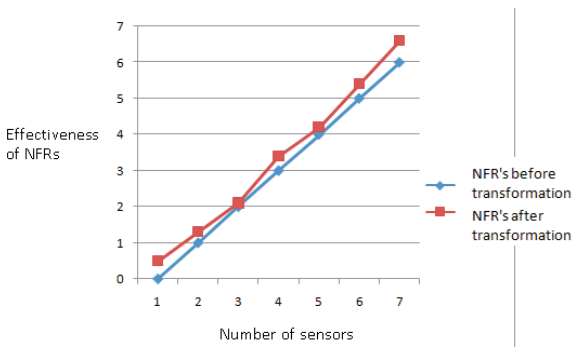

Figure 12. Comparison of NFR identification using evaluation and the transformation

Our work is related to several others in research activities. Several design methodologies have already defined by many researchers. For evaluating the architecture we have several methods such as SAAM, ATAM etc., which all concentrates on the evaluation of NFRs based on the scenarios, where we have additionally used mathematical modelling, simulation, and reasoning. Research on the metrics were implemented on the systems after the development, whereas we have worked in the early stage of the design and deployed using the design. Several research communities work on the NFRs using object oriented design methodologies which concentrate on reusability and maintenance. Real time system design has also been concentrated towards the identification of NFR's. The method which we have proposed in this paper is different, and we have concentrated on the realistic situation and addressed the system NFRs which needs to be balanced. In our method we have used some transformations to improve the architecture and find the NFRs; the transformation doesn't mean the conversion of architecture. We have tried to verify the transformations with the help of UML 4.2.1 and we have compared with one of the normal method to identify the NFRs.

CONCLUSION

The architectural design method presented here directly handles non-functional requirements put on the architecture. We have also identified that the capability of fulfilling a few of the non-functional requirements is largely dependent on the architecture that is being used in the first place.

We begin by the first iteration in which only the functional requirements are taken into account. The next iterations focus on evaluating the architecture for the NFRs and transforming it to better fulfilment. NFRs may be evaluated using scenario based evaluation, simulation methods, mathematical modelling and reasoning whereas the transformations have been done by

1.Impose the architectural style.
2.Impose the architectural pattern.
3.Impose a design pattern.
4.NFRs to the functionality conversion.
5.Requirements distribution.

We implemented the method on the SmartHome application. The experimented results are shown in the section 7, which has carried out an appreciable support for the engineers and researchers who work on the architectural design.

REFERENCES

I.Pratima Singh and Anil Kumar Tripathi, Issues in Testing of Software with NFR, International Journal of Software Engineering & Applications (IJSEA), Vol.3, No.4, July 2012

II.Ameller, D,et.al., Dealing with Non-Functional Requirements in Model-Driven Development , Engineering Conference (RE),ieeexplore.ieee.org, 2010.

III.Mehta. R, Tomás Ruiz-López, L.Chung and M. Noguera, Selecting among alternatives using dependencies: an NFR approach, SAC 13,Proceedings of the 28th Annual ACM Symposium on Applied Computing Pages 1292-1297,2013.
IV.Yi Liu ,Zhiyi Ma , Rui Qiu, Hongjie Chen and Weizhong Shao, An Approach to Integrating Non-functional Requirements into UML Design Models Based on NFR-Specific Patterns, Quality Software (QSIC), 12th International Conference, IEEExplore digital library, 2012.
V.Filieri.A. et.al, A formal approach to adaptive software: continuous assurance of non-functional requirements,Formal Aspects of Computing,Springer-, 2012.
VI.Surpass Series Products, Siemens Information and Comm. Network,www.siemens.com/surpass, 2007.
VII.Cortellessa. V., P. Pierini, and D. Rossi, On the Adequacy of UML-RT for Performance Validation of an SDH Telecommunication System, Proc. Int'l Symp. Object-Oriented Real-Time Distributed Computing, 2005.
VIII.Umar. M, M. N. Ahmedkhan, A Framework to Separate Non-Functional Requirements for System Maintainability, Kuwait journal of Science and Engineering, 39 (1B) pp.211 - 231, 2012.
IX.Romina Ermo, Vittorio Cortellessa, Alfonso Pierantonio, Michele Tucci., "Performance Driven architectural refactoring through bidirectional model Transformation", ACM SIGSOFT Conference on Quality of Software Architectures, QoSA2012.
X.Aroul canessane. R, S. Srinivasan, A Framework for Analysing the System Quality, ieeexplore.ieee.org, ICCPCT 2013.
XI.Krogstie. J, Sindre. G, and Jørgensen. H, "Process Models Representing Knowledge for Action: A Revised Quality Framework," European J. Information Systems, vol. 15, no. 1, pp. 91-102, 2006.
XII.Aroul canessane. R, S. Srinivasan, UML Model Transformation for a Product Line Design, International Journal of Engineering & Technology (IJET), 2013.
XIII.Aldrich. J, "Using Types to Enforce Architectural Structure," Proc. Working IEEE/Int'l Federation for Information Processing (IFIP) Conf.Software Architecture, pp. 211-220, 2008.
XIV.Casamayor. A, D Godoy, M Campo, Identification of non-functional requirements in textual specifications: A semi-supervised learning approach, Information and Software Technology, Elsevier 2010.

R. AROUL CANESSANE
Research Scholar, Sathyabama University, Chennai